

## Звіт

### про виконання лабораторної роботи №2

#### з дисципліни

#### «Розпізнавання образів»

Виконали: студенти IV курсу

групи КА-76

Панасюк Я.І.

та групи КА-76

Іванов С.

Перевірила:

Дідковська М.В.

Київ – 2020

## Завдання

Кожен учасник або учасниця команди спершу обирає дескриптор (один із розглянутих у лекціях або ж знайдений окремо) та предмет на прикладі якого відбуватиметься дослідження. Враховуючи що вас багато, будь ласка обирайте унікальніші предмети за улюблену чашку/телефон/мишку. Маючи те і інше наготові кожен учасник бригади має зняти не менше сотні фото предмета, варіючи його розміщення та ракурс в кадрі, освітлення, наявність візуальних перешкод, зашкапеність зображення, фокусну віддаль та тремтіння рук. Сотня фото обраного предмету на однаковій сцені з однаковою якістю зйомки, але з різних ракурсів на жаль не підійде, постарайтесь наполегливо варіювати сцену і умови зйомки. До цих фото варто додати невелику підбірку зображень, що не містять предмет, або ж містять предмет візуально подібний до вашого, штук 20 повинно вистачити, якщо залишитись натхнення можна й більше. Після чого ми нарешті дійшли до цікавого, а саме до дослідження: Вам потрібно згенерувати обраний дескриптор для обраного предмета, після чого з його допомогою розпізнати об'єкт на всій тестовій вибірці збираючи при цьому такі метрики: відносна кількість правильно сумішених ознак, похибка локалізації (відстань між реальним розміщенням предмета в кадрі та розпізнаним) та відносний час обробки фото в залежності від розміру зображення. Метрики мають зберегтись у файлику для подальших досліджень. Наступним кроком ви обмінюєтесь об'єктом з колегою, і уже маючи готову збіралку метрик, обчислюєте їх для предмета вашого сусіда, таким чином у вас збирається 9 наборів даних, по три на дескриптор. Самою ж ідеєю лаби є дослідити розбіжності у роботі ваших дескрипторів та виконати порівняльний аналіз їх поведінки, сформулювати висновки з викладками і прикладами так аби було зрозуміло вам та, сподіваюсь, усім вашим колегам. Таким чином кінцевим результатом буде від вас гуглдок з описом виняткових особливостей, сильних та слабких сторін дескриптора і обґрунтуванням чому вони поводяться саме так.

## Хід роботи

### Вибіркі об'єктів

```
In [1]: import glob
import cv2

#DUCK_DS
duck_train_pth = glob.glob("duck_ds/train/*.jpg")
duck_test_pth = glob.glob("duck_ds/test/*.jpg")
duck_ds_train = [cv2.imread(_) for _ in duck_train_pth]
duck_ds_test = [cv2.imread(_) for _ in duck_test_pth]
reduced_duck_ds_train = [cv2.resize(cv2.imread(_), (0,0), fx=0.5, fy=0.5) for _ in duck_train_pth]
reduced_duck_ds_test = [cv2.resize(cv2.imread(_), (0,0), fx=0.5, fy=0.5) for _ in duck_test_pth]

#CUP_DS
cup_train_pth = glob.glob("cup_ds/train/*.jpg")
cup_test_pth = glob.glob("cup_ds/test/*.jpg")
cup_ds_train = [cv2.imread(_) for _ in cup_train_pth]
cup_ds_test = [cv2.imread(_) for _ in cup_test_pth]
reduced_cup_ds_train = [cv2.resize(cv2.imread(_), (0,0), fx=0.5, fy=0.5) for _ in cup_train_pth]
reduced_cup_ds_test = [cv2.resize(cv2.imread(_), (0,0), fx=0.5, fy=0.5) for _ in cup_test_pth]
```

### Дескриптори

ORB (Oriented FAST and Rotated BRIEF)

```
In [2]: import numpy as np
import cv2
import time

def ORB_descript(query_img, train_img, show_final_img=False):
    # Read the query image as query_img
    # and traing image This query image
    # is what you need to find in train image

    # Convert it to grayscale
    query_img_bw = cv2.cvtColor(query_img,cv2.COLOR_BGR2GRAY)
    train_img_bw = cv2.cvtColor(train_img, cv2.COLOR_BGR2GRAY)

    # Initialize the ORB detector algorithm
    orb = cv2.ORB_create()

    t_start = time.time()

    # Now detect the keypoints and compute
    # the descriptors for the query image
    # and train image
    queryKeypoints, queryDescriptors = orb.detectAndCompute(query_img_bw,None)
    trainKeypoints, trainDescriptors = orb.detectAndCompute(train_img_bw,None)

    if(queryDescriptors is None or trainDescriptors is None):
        return 0, 966, 0

    # Initialize the Matcher for matching
    # the keypoints and then match the
    # keypoints
    matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = matcher.match(queryDescriptors,trainDescriptors)

    t_end = time.time()
    rel_time = t_end - t_start

    # Sort them in the order of their distance.
    matches = sorted(matches, key = lambda x:x.distance)

    if(show_final_img):
        # draw the matches to the final image
        # containing both the images the drawMatches()
        # function takes both images and keypoints
        # and outputs the matched query image with
        # its train image
        final_img = cv2.drawMatches(query_img, queryKeypoints,
            train_img, trainKeypoints, matches[:20],None)

        final_img = cv2.resize(final_img, (1000,650))

        # Show the final image
        cv2.imshow("Matches", final_img)
        cv2.waitKey(3000)

    # Metrics
    # the relative number of correctly matched features
    rel_num = len(matches)*2 / (len(queryDescriptors) + len(trainDescriptors))

    # localization inaccuracy
    distances = [_distance for _ in matches]
    avr_dist = sum(distances)/len(distances)

    return rel_num, avr_dist, rel_time
```

BRIEF\_descript

```
In [3]: import numpy as np
import cv2
import time

def BRIEF_descript(query_img, train_img, show_final_img=False):
    # Read the query image as query_img
    # and train image This query image
    # is what you need to find in train image

    # Convert it to grayscale
    query_img_bw = cv2.cvtColor(query_img,cv2.COLOR_BGR2GRAY)
    train_img_bw = cv2.cvtColor(train_img, cv2.COLOR_BGR2GRAY)
    #-----
    # Initiate FAST detector
    star = cv2.xfeatures2d.StarDetector_create()
    # Initiate BRIEF extractor
    brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()

    t_start = time.time()

    # Now detect the keypoints and compute
    # the descriptors for the query image
    # and train image
    # find the keypoints with STAR
    queryKeypoints = star.detect(query_img_bw,None)
    trainKeypoints = star.detect(train_img_bw,None)
    # compute the descriptors with BRIEF
    queryKeypoints, queryDescriptors = brief.compute(query_img_bw, queryKeypoints)
    trainKeypoints, trainDescriptors = brief.compute(train_img_bw, trainKeypoints)

    #-----
    if(queryDescriptors is None or trainDescriptors is None):
        return 0, 966, 0

    # Initialize the Matcher for matching
    # the keypoints and then match the
    # keypoints
    matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = matcher.match(queryDescriptors,trainDescriptors)

    t_end = time.time()
    rel_time = t_end - t_start

    # Sort them in the order of their distance.
    matches = sorted(matches, key = lambda x:x.distance)

    if(show_final_img):
        # draw the matches to the final image
        # containing both the images the drawMatches()
        # function takes both images and keypoints
        # and outputs the matched query image with
        # its train image
        final_img = cv2.drawMatches(query_img, queryKeypoints,
            train_img, trainKeypoints, matches[:20],None)

        final_img = cv2.resize(final_img, (1000,650))

        # Show the final image
        cv2.imshow("Matches", final_img)
        cv2.waitKey(3000)

    # Metrics
    # the relative number of correctly matched features
    rel_num = len(matches)*2 / (len(queryDescriptors) + len(trainDescriptors))

    # localization inaccuracy
    distances = [_distance for _ in matches]
    avr_dist = sum(distances)/len(distances)

    return rel_num, avr_dist, rel_time
```

### Метрики

```
In [4]: import matplotlib.pyplot as plt
from bokeh.plotting import figure, output_file, show
from pathlib import Path

def get_metrics(descriptor, ds_train, ds_test, ds_name, descr_name, plotting_graphs=True):
    """It creates files with "the relative number of correctly matched features" and "localization inaccuracy" metrics"""

    rel_num_of_correct_mathced_feat=[]
    distances=[]
    relative_time=[]

    for train_img in ds_train:
        distances_to_train_img =[]
        rel_num_for_train_img =[]
        time_for_train_img =[]

        for test_img in ds_test:
            rel_num, avr_dist, time = descriptor(train_img, test_img)

            rel_num_for_train_img.append(rel_num)
            distances_to_train_img.append(avr_dist)
            time_for_train_img.append(time)

        rel_num_of_correct_mathced_feat.append(rel_num_for_train_img)
        distances.append(distances_to_train_img)
        relative_time.append(time_for_train_img)

    # Create dirs for output data
    Path(f'{ds_name}/metrics/{descr_name}/data').mkdir(parents=True, exist_ok=True)
    Path(f'{ds_name}/metrics/{descr_name}/graphs').mkdir(parents=True, exist_ok=True)

    # write metrics for every train img
    for i in range(len(ds_train)):
        with open(f'{ds_name}/metrics/{descr_name}/data/train_img_{i}_rel_num.txt", "w") as file1:
            file1.write("\n".join([str(_) for _ in rel_num_of_correct_mathced_feat[i]]))
        with open(f'{ds_name}/metrics/{descr_name}/data/train_img_{i}_avr_dist.txt", "w") as file2:
            file2.write("\n".join([str(_) for _ in distances[i]]))
        with open(f'{ds_name}/metrics/{descr_name}/data/train_img_{i}_relative_time.txt", "w") as file3:
            file3.write("\n".join([str(_) for _ in relative_time[i]]))

    if(plotting_graphs):
        arr=[_ for _ in range(len(ds_test))]

        for i in range(len(ds_train)):
            output_file(f'{ds_name}/metrics/{descr_name}/graphs/train_img_{i}_rel_num.html")
            p = figure(title=f"the relative number of correctly matched features from train_img_{i}" ,
                x_axis_label='test_img_id', y_axis_label='rel num')
            p.line(arr, rel_num_of_correct_mathced_feat[i], legend_label="Temp.", line_width=2)
            show(p)

        for i in range(len(ds_train)):
            output_file(f'{ds_name}/metrics/{descr_name}/graphs/train_img_{i}_distance_graph.html")
            p = figure(title=f"Distance to train_img_{i}" , x_axis_label='test_img_id', y_axis_label='distance')
            p.line(arr, distances[i], legend_label="Temp.", line_width=2)
            show(p)

        for i in range(len(ds_train)):
            output_file(f'{ds_name}/metrics/{descr_name}/graphs/train_img_{i}_relative_time_graph.html")
            p = figure(title=f"Relative time for train_img_{i}" , x_axis_label='test_img_id', y_axis_label='relative time')
            p.line(arr, relative_time[i], legend_label="Temp.", line_width=2)
            show(p)
```

### Отримання метрик

```
In [10]: get_metrics(ORB_descript, duck_ds_train, duck_ds_test, "duck_ds", "ORB")

In [6]: get_metrics(ORB_descript, cup_ds_train, cup_ds_test, "cup_ds", "ORB")

In [7]: get_metrics(BRIEF_descript, cup_ds_train, cup_ds_test, "cup_ds", "BRIEF")

In [9]: get_metrics(BRIEF_descript, duck_ds_train, duck_ds_test, "duck_ds", "BRIEF")

In [ ]: get_metrics(ORB_descript, reduced_duck_ds_train, reduced_duck_ds_test, "duck_ds", "reduced_ORB")

In [ ]: get_metrics(ORB_descript, reduced_cup_ds_train, reduced_cup_ds_test, "duck_ds", "reduced_ORB")

In [ ]: get_metrics(BRIEF_descript, reduced_cup_ds_train, reduced_cup_ds_test, "cup_ds", "reduced_BRIEF")

In [ ]: get_metrics(BRIEF_descript, reduced_duck_ds_train, reduced_duck_ds_test, "duck_ds", "reduced_BRIEF")
```

### Результати

Отримані метрики можна побачити у файлах [Назва датасету]\_[Метрика]

Графіки на основі отриманих даних [Назва датасету]\_[Метрика]\_[Номер тренованого зображення в масиві]

Щоб вивести дескпиптор конкретного зображення, введіть номер тренованого й тестового зображення у масиві

```
In [ ]: rel_num, avr_dist, rel_time = ORB_descript(duck_ds_train[1], duck_ds_test[40], show_final_img=True)
```

### Аналіз